CIÊNCIA DA COMPUTAÇÃO

4º Período

Desenvolvimento de sistema para análise de performance de algoritmos de ordenação de dados

Nome: Rodrigo Joaquim Sousa de Lima

RA: F32IEA0

Turma: CC4A30

Brasília, 18 de novembro de 2021

Índice

1. Introdução

Neste projeto, discutiremos algoritmos de ordenação, o algoritmo de ordenação mais próximo da estrutura de uma linguagem estruturada, que permite organizar listas de números ou palavras de acordo com as necessidades do usuário.

Antes de discutir algoritmos de classificação, devemos primeiro entender o que é um algoritmo e de onde ele vem. A palavra algoritmo está relacionada ao nome do matemático Abu Abdullah Muhammad Bin Musa AlKhwarizmique, que viveu entre os séculos VIII e IX. Seu trabalho era preservar e disseminar o conhecimento da Grécia e da Índia antigas. Seus livros são fáceis de entender porque sua principal filosofia e valor não é criar teoremas e correntes de pensamento, mas simplificar a matemática a um nível de compreensão acessível a todos. (História do algoritmo: os primeiros passos da computação. Test Online, 2021. Disponível em: http://www.testonline.com.br/historia-do-algoritmo-os-primeiros-passos-da-computacao/, acesso em 28 de outubro de 2021)

O conceito de algoritmo remonta a vários séculos e seu uso remonta a matemáticos russos, como a peneira de Eratóstenes e o algoritmo euclidiano.

Os conceitos de algoritmos são frequentemente ilustrados por fórmulas de exemplo, embora muitos algoritmos sejam mais complexos. Eles podem repetir etapas (iteração) ou tomar decisões rápidas (como comparações ou lógica) até que a tarefa seja concluída. Um algoritmo implementado corretamente não resolverá um problema se for mal implementado ou se não corresponder ao problema de Jean Luc Chabert.

Um algoritmo não representa necessariamente um programa de computador, mas sim as etapas necessárias para completar uma tarefa. Sua implementação pode ser realizada por um computador, por outro tipo de autômato ou por um ser humano. Algoritmos diferentes podem executar a mesma tarefa usando um conjunto diferente de instruções com mais ou menos tempo, espaço ou esforço do que outros algoritmos. Essa diferença pode refletir a complexidade computacional aplicada, que depende da estrutura de dados completa do algoritmo. Por exemplo, um algoritmo de vestir pode especificar que você coloque as meias e os sapatos antes de colocar as calças, enquanto outro especifica que você coloque as calças primeiro, depois as meias e os sapatos.

Obviamente, o primeiro algoritmo é mais difícil de implementar do que o segundo, mesmo que ambos conduzam ao mesmo resultado.

Em matemática e ciência da computação, um algoritmo é uma sequência finita de ações possíveis para encontrar uma solução para um tipo específico de problema. De acordo com Dasgupta, Papadimitriou e Vazirani; “Algoritmos são procedimentos precisos, inequívocos, padronizados, eficientes e precisos.”

Suas características são: acabado, o algoritmo pode ter que resolver o problema; bem definidas: as etapas devem ser definidas de forma que possam ser compreendidas; (Algoritmo, Wikipedia, 2018. Disponível em: https://pt.wikipedia.org/wiki/Algoritmo, acesso em 30 de outubro de 2021.)

Para sermos eficazes, devemos sempre resolver os problemas que temos para resolver e antecipar as falhas. Assim, graças à especificação clara e concisa de como calcular sistematicamente, algoritmos podem ser definidos em dispositivos mecânicos semelhantes ao ábaco.

Augusta Ada Byron King, condessa de Lovelace, agora conhecida como Ada Lovelace, era uma matemática e escritora inglesa. Hoje, ela é mais conhecida por ter escrito o primeiro algoritmo a ser processado por máquina, Charles Babbage's Analyzer, que em sua pesquisa detalhou como o Analyzer funcionava em 1842, quando publicou seu artigo científico ela fez comentários proféticos. Isso incluía suas previsões de uma máquina que poderia ser usada para compor música complexa, gráficos complexos, produzidos e usados para fins científicos e práticos. Portanto, é graças ao seu trabalho que os historiadores a reconhecem como a primeira programadora. (História do algoritmo: os primeiros passos da computação. Test Online, 2021. Disponível em: http://www.testonline.com.br/historia-do-algoritmo-os-primeiros-passos-da-computacao/. Acesso em 30 de outubro de 2021)

O algoritmo é um mecanismo que é cego e não quer seguir um conjunto de regras que são sistematicamente aplicadas a uma entrada apropriada e resolve seus problemas dentro de um período de tempo finito. Portanto, um algoritmo é como uma “receita” para realizarmos uma tarefa ou resolvermos um problema. Se seguirmos uma receita de bolo corretamente poderemos fazer o bolo e como qualquer receita um algoritmo deve seguir certas instruções para chegar à solução do assunto. (Na verdade, o

que [...] é exatamente um algoritmo? ElPaís, 2018. Disponível em: https://brasil.elpais.com/brasil/2018/03/30/tecnologia/1522424604_741609.html. Acesso em 01 de novembro de 2021)

Um desenvolvedor, ao construir um algoritmo, deve antecipar todos os tipos de cenários possíveis ao executar o algoritmo e saber como criar estratégias para decompor problemas do mundo real em instruções abstratas mais imaginativas para que o computador possa seguir a solução de um problema. (O que é algoritmo? Entenda como funciona em apps e sites da Internet. Techtudo, 2020. Disponível em: https://www.techtudo.com.br/listas/2020/05/o-que-e-algoritmo-entenda-como-funciona-em-apps-e-sites-da-internet.ghtml, Acesso em 01 de novembro de 2021.)

As linguagens de programação já possuem métodos de classificação, mas é bom saber como os algoritmos funcionam, pois existem alguns problemas que os algoritmos de classificação geral não resolvem, às vezes exigindo ajustes. Pode ser implementado com qualquer sequência de valores ou objetos com lógica infinita (por exemplo, língua portuguesa, linguagem Pascal, linguagem C, uma sequência de números, um conjunto de objetos como uma caneta, um lápis e borracha), ou seja, qualquer coisa que pode fornecer uma sequência lógica. (Algoritmos de ordenação: análise e comparação. Devmedia, 2013. Disponível em: https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261. Acesso em 01 de novembro de 2021.)

Os algoritmos de classificação mais comuns são: inserir classificação, selecionar classificação, bolha classificar, combinar classificação, classificação rápida, mesclar classificação, classificação de heap e classificação de shell. Neste projeto, algoritmos como classificação por bolha, classificação selecionada, classificação por inserção, classificação rápida, classificação por mesclagem e classificação por heap serão apresentados.

2. Referencial teórico

O objetivo da ordenação é facilitar a recuperação dos dados de uma lista, o algoritmo de ordenação coloca os elementos de uma dada sequência em uma certa ordem e cada tipo de algoritmo de ordenação tem o seu jeito de executar.

2.1. Bubble sort

Bubble Sort é o algoritmo mais simples, mas menos eficiente. Nesse algoritmo, cada elemento da posição i será comparado ao elemento da posição i + 1, ou seja, um elemento da posição 2 será comparado ao elemento da posição 3 da posição 2 é maior do que aquele da posição 3, eles troque de lugar e assim por diante. Devido a essa implementação, o vetor terá que ser percorrido quantas vezes forem necessárias, tornando o algoritmo ineficiente para listas muito grandes.

Para entender o Bubble sort, considere uma matriz não classificada [1, 23, 10, 2] e discutir cada etapa realizada para classificar a matriz em ordem crescente. Em cada estágio, dois itens adjacentes são verificados e trocados se encontrados na ordem errada.

Primeiro passo: (1) e (23) são comparados e encontrados na ordem correta (ordem crescente neste caso), após o qual (23) e (10) são comparados, uma vez que (23> 10), então, esses números são trocados Em seguida, (23) e (2) são comparados e trocados.

Segundo passo: (1) e (10) são comparados e estão na ordem certa, então (10) e (2) são comparados, já que (10> 2), portanto, esses números estão invertidos. Em seguida, (10) e (23) são comparados e encontrados na ordem correta.

Terceiro passo: (1) e (2) são comparados, visto que (1> 2), portanto, esses números são trocados, então (1,10) e (10,23) são verificados e encontrados na ordem correta. (PHP Program – Bubble Sort. AlphaCodingSkill, 2018. Disponível em: https://www.alphacodingskills.com/php/pages/php-program-for-bubble-sort.php, Acesso em 02 de novembro de 2021)

2.2. Selection sort

O Selection sort baseia-se na passagem sempre do menor valor do vetor para a primeira posição (ou o maior de acordo com a ordem solicitada), depois o segundo menor valor para a segunda posição e assim por diante, até o último dois elementos.

Neste algoritmo de ordenação, um número é escolhido a partir do primeiro, este número escolhido é comparado com os números à sua direita, quando um número menor é encontrado, o número escolhido ocupa a posição do menor número encontrado. Se não for encontrado um número menor que o escolhido, ele é colocado no lugar do primeiro número escolhido, e o próximo número à sua direita será escolhido para fazer as comparações. Este processo é repetido até que a lista seja classificada.

Para entender a classificação da seleção, vamos considerar uma matriz não classificada [1, 10, 23, -2] e discutir cada etapa realizada para classificar a matriz em ordem crescente. Em cada passagem, o menor elemento é encontrado no array não classificado e trocado pelo primeiro elemento do array não classificado.

Primeira passagem: A matriz inteira é uma matriz não classificada e (-2) é o menor número na matriz. Depois de encontrar (-2) como o menor número, ele é trocado pelo primeiro elemento da matriz.

Segunda passagem: neste exemplo, a matriz do lado esquerdo é a matriz classificada e o comprimento dessa matriz aumenta em um após cada iteração. Após a primeira passagem, o comprimento da matriz classificada é um. O resto da matriz é uma matriz não classificada. (1) é o menor número no array não classificado que é trocado pelo primeiro elemento do array não classificado. Após esta troca, o comprimento do array ordenado e do array não ordenado será dois.

Terceira passagem: (10) é o menor número no array não classificado e trocado pelo primeiro elemento do array não classificado.

Quarta passagem: (23) é o único número na matriz não classificada e está na posição correta. (PHP Program – Selection Sort. AlphaCodingSkill, 2018. Disponível em: https://www.alphacodingskills.com/php/pages/php-program-for-selection-sort.php. Acesso em 02 de novembro de 2021)

2.3. Insertion sort

O Insertion sort é baseado na ideia de consumir um elemento da matriz não classificada e colocá-lo na posição correta na matriz classificada. Isso fará com que o

comprimento da matriz classificada aumente em um e o comprimento da matriz não ordenada diminuirá em um após cada iteração.

Entendendo o Insertion Sort, vamos considerar um array não classificado [23, 1, 10, 5, 2] e discutir o passo a passo para ordenar o array em ordem crescente. Em cada passagem, um elemento é retirado da matriz não classificada e inserido na posição correta na matriz classificada.

Primeiro passo: o array inteiro é um array não classificado e (23) é considerado inserido no array ordenado. Como (23) é o primeiro elemento da matriz classificada e não tem elementos de comparação, ele permanece no lugar. Segundo passo: (1) é extraído do array não classificado para colocá-lo no array ordenado, comparado a todos os elementos do array ordenado e descobre que (23) é o único número maior que (1). (1) é inserido na matriz classificada e a posição de (23) é deslocada em uma posição na matriz, de modo que o comprimento da matriz classificada aumenta em um e o comprimento da matriz não classificada diminui em um.

Terceiro passo: (10) é extraído do array não classificado e comparado com todos os elementos do array ordenado. (23) é o único número na matriz ordenada maior que (10). (10) é inserido na matriz ordenada e a posição de (23) é movida para uma posição na matriz.

Quarto passo: (5) é comparado a todos os elementos do array ordenado e verifica-se que (10) e (23) são os números que precisam ser movidos para um ponto para inserir (5) no array ordenado.

Quinto passo: Para inserir (2) na matriz ordenada, (5), (10) e (23) são movidos para um ponto. (PHP Program – Insertion Sort. AlphaCodingSkill, 2018. Disponível em: https://www.alphacodingskills.com/php/pages/php-program-for-insertion-sort.php. Acesso em 02 de novembro de 2021).

2.4. Quick sort

O Quick sort é o algoritmo de ordenação por comparação mais eficiente, no qual um elemento denominado pivô é escolhido, a partir do qual a lista é organizada de forma que todos os números anteriores sejam menores e todos os números posteriores sejam

maiores. Ao final desse processo, o número do pivô já está em sua posição final. Ambos os grupos codificados recursivamente passaram pelo mesmo processo até que a lista fosse classificada.

Para entender a classificação rápida, considere uma matriz não classificada [4, 1, 25, 50, 8, 10, 23] e discuta cada etapa realizada para classificar a matriz em ordem crescente.

Na implementação deste exemplo, o último elemento da matriz é escolhido como o elemento pivô. No início do processo de partição, as variáveis iej são criadas, que são inicialmente iguais. Conforme a pontuação avança, os valores de i e j tornam-se diferentes. i significa o limite entre os elementos pivô menores e os elementos pivô maiores. j indica a fronteira entre os principais elementos do pivô e a matriz não particionada.

Após o particionamento, ele gera duas partições com a partição esquerda contendo elementos menores que o pivô e a partição direita contendo elementos maiores que o pivô.Ambas as partições são reparticionadas com a mesma lógica e esse processo continua até que uma partição tenha zero ou um elemento. O resultado final deste processo será uma matriz classificada. (PHP Program – Quick Sort. AlphaCodingSkill, 2018. Disponível em: https://www.alphacodingskills.com/php/pages/php-program-for-quick-sort.php. Acesso em 02 de novembro de 2021)

2.5. Merge sort

O Merge sort é um exemplo de algoritmo de classificação para a comparação de divisão e conquista.

Sua ideia básica é Dividir (o problema em vários subproblemas e resolver esses subproblemas recursivamente) e Conquistar (uma vez que todos os subproblemas são resolvidos, a conquista ocorre, ou seja, a união das resoluções de subproblemas). alto consumo de memória e tempo de execução, tornando esta técnica ineficiente em alguns problemas.

As três etapas úteis dos algoritmos de divisão e conquista, que se aplicam à classificação por mesclagem, são:

- Divide: calcula o ponto médio do subarray, que leva um tempo constante 0 (1);
- Conquista: resolve recursivamente dois subproblemas, cada um de tamanho n / 2, que contribuem com 2T (n / 2) para a execução;
- Combinar: Unir os sub-arranjos em um único conjunto ordenado, que leva o tempo 0(n);

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 2T(n/2) + \Theta(n), & \text{se } n > 1 \end{cases}$$

Complexidade

Complexidade de tempo: 0 (nlog2n)

Complexidade de espaço: 0(n)

(PHP Program – Merge Sort. AlphaCodingSkill, 2018. Disponível em: https://www.alphacodingskills.com/php/pages/php-program-for-merge-sort.php. Acesso em 02 de novembro de 2021 e Merge sort. Wikipedia, 2021. Disponível em: https://pt.wikipedia.org/wiki/Merge_sort. Acesso em 03 de novembro de 2021)

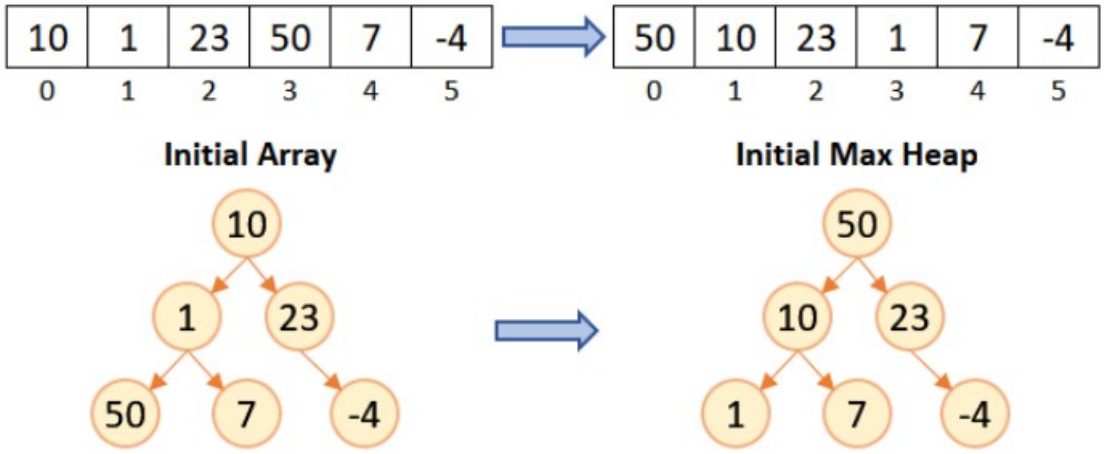
2.6. Heap sort

O Heap Sort usa a propriedade heap para classificar a matriz. Isso é para criar um heap máximo para aumentar a classificação que contém o maior elemento de heap na raiz. Para classificar em ordem decrescente, heap mínimo é usado, que contém o menor item de heap na raiz. A etapa de classificação em ordem crescente de classificação de heap é resumida abaixo:

- Etapa 1: Cria um heap máximo que contém o maior elemento de heap na raiz;
- Etapa 2: Troca o último elemento no heap pelo elemento raiz e remova o último elemento do heap. Com os itens restantes, repite a etapa 1.

Para entender a classificação de heap, considere uma matriz não classificada [10, 1, 23, 50, 7, 4] e discuta cada etapa realizada para classificar a matriz em Ascendente.

A figura a seguir mostra a estrutura de heap da matriz de entrada e o heap máximo.O número do índice do elemento raiz do heap é 0. No heap máximo, o elemento no heap maior sempre reside na raiz.



Depois de construir o heap máximo inicial, o último elemento do heap é substituído pelo elemento raiz e o último elemento que contém o maior número na matriz é removido do heap. Os itens restantes na pilha são retomados através da função heapfy onde é fornado uma pilha máxima e o número de itens será reduzido em 1. Esse processo continua até que o número de itens na pilha seja 1. Nesse ponto, a matriz será classificada. (PHP Program – Heap Sort. AlphaCodingSkill, 2018. Disponível em: https://www.alphacodingskills.com/php/pages/php-program-for-heap-sort.php. Acesso em 03 de novembro de 2021)

3. Desenvolvimento

O código fonte foi desenvolvido em PHP e através da linha de comando o usuário interage com o programa, informando parâmetros que são obrigatórios durante execução:

nome do programa PHP: execute_sorts.php.

1º Caso – Executando apenas um tipo de ordenação

Ao executar o comando na pasta onde se encontram os arquivos de código fonte e o arquivo csv que será utilizado durante a execução, o programa apresentará na linha de comando uma mensagem para o usuário interagir. No exemplo a seguir a primeira opção é o número de linhas:

```
Digite o nº de linhas que serão lidas do arquivo :
```

Caso o usuário informe um valor menor ou igual a zero ou um valor inválido, o programa informa uma mensagem de erro e solicita que o usuário informe número de linhas novamente:

```
Digite o nº de linhas que serão lidas do arquivo : 0
você informou um nº de linhas inválido!
Digite o nº de linhas que serão lidas do arquivo :
```

A seguir, é requisitado pelo programa que o usuário digite um número, para ser armazenado e depois usado para a ordenação.

```
Digite o tipo de organização que será(ão) realizado(s) :  1 - Buble Sort, 2 - Selection Sort, 3 - Ins
ertion Sort,     4 - Quick Sort, 5 - Merge Sort, 6 - Heap Sort, 7 - Todos:
```

Caso o usuário informe um valor diferente dos números apresentados, o programa informa uma mensagem de erro e solicita que o usuário informe o tipo de organização novamente:

```
Digite o tipo de organização que será(ão) realizado(s) :  1 - Buble Sort, 2 - Selection Sort, 3 - In2
123
você informou uma opção inválida!
Digite o tipo de organização que será(ão) realizado(s) :  1 - Buble Sort, 2 - Selection Sort, 3 - Ins
ertion Sort,     4 - Quick Sort, 5 - Merge Sort, 6 - Heap Sort, 7 - Todos:
```

No exemplo abaixo, informamos o número de 12.000 linhas e a opção 1 (Bubble Sort). Neste momento, o programa executa o tipo de organização escolhida e o número de linhas que será lido do arquivo de entrada (Metadados_APS.csv)

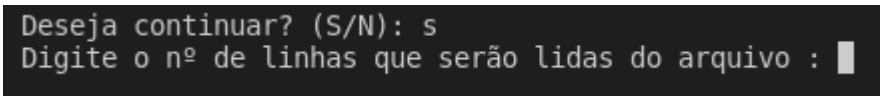
```
Digite o tipo de organização que será(ão) realizado(s) :  1 - Buble Sort, 2 - Selection Sort, 3 - Ins
ertion Sort,     4 - Quick Sort, 5 - Merge Sort, 6 - Heap Sort, 7 - Todos: 1
iniciando o Bubble Sort...
Tempo total de execução de 12000 linhas após o bubble sort foi de : 4.4920 segundos

Memória usada: 6.2730 MegaBytes
```

Ao final da execução, o programa exibe uma mensagem com o resultado. No exemplo acima o resultado foi realizado nas 12.000 linhas, com tempo de 4,4920 segundos. O programa também informa a quantidade de memória utilizada. No exemplo acima foi de 6.2730 MegaBytes.

Após exibir o relatório fina de execução, o usuário tem a opção de continuar ou finalizar o programa, conforme exemplo abaixo:

```
Deseja continuar? (S/N):
```

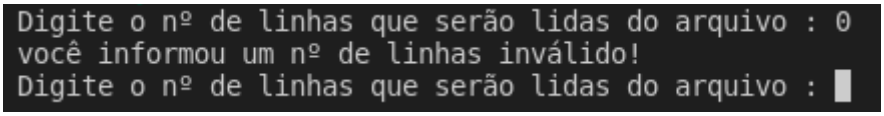
Caso o usuário selecione a opção ‘N’, o programa é encerrado. Caso seja escolhida seja a opção ‘S’, o programa é reiniciado e informa ao usuário as opções iniciais:



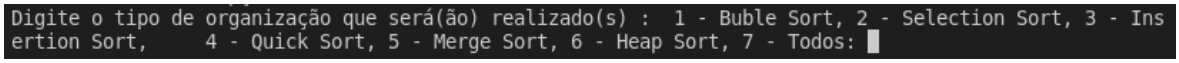
2° Caso – Executando todos os tipos de organizações possíveis

Ao executar o comando na pasta onde se encontram os arquivos de código fonte e o arquivo csv que será utilizado durante a execução, o programa apresentará na linha de comando uma mensagem para o usuário interagir. No exemplo a seguir a primeira opção é o número de linhas:

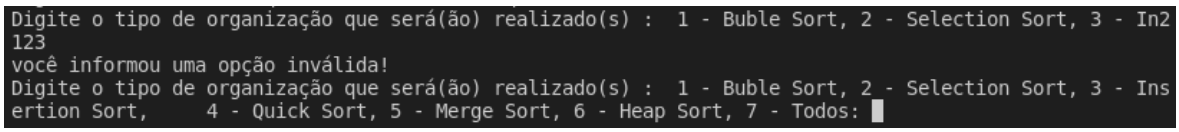
Caso o usuário informe um valor menor ou igual a zero ou um valor inválido, o programa informa uma mensagem de erro e solicita que o usuário informe número de linhas novamente:



A seguir, é requisitado pelo programa que o usuário digite um número, para ser armazenado e depois usado para a ordenação.



Caso o usuário informe um valor diferente dos números apresentados, o programa informa uma mensagem de erro e solicita que o usuário informe o tipo de organização novamente:



No exemplo abaixo, informamos o número de 12.000 linhas e a opção 7 (Todos). Neste momento, o programa executa o tipo de organização escolhida e o número de linhas que será lido do arquivo de entrada (Metadados_APS.csv)

Ao final da execução, o programa exibe uma mensagem com o resultado. No exemplo acima o resultado foi realizado nas 12.000 linhas, com os tempos de cada um dos tipos de organização possíveis, no formato de ranking das execuções. O programa também informa a quantidade de memória utilizada. No exemplo acima foi de 37.6348 MegaBytes.

A apresentação dos resultados comparativos entre os métodos e diferentes casos estará contida no próximo tópico:

4. Resultados e discussão

Os resultados que serão exibidos a seguir foram obtidos através de um computador com processador Dual–Core 2.5GHz e memória RAM DDR3 de 8GB.

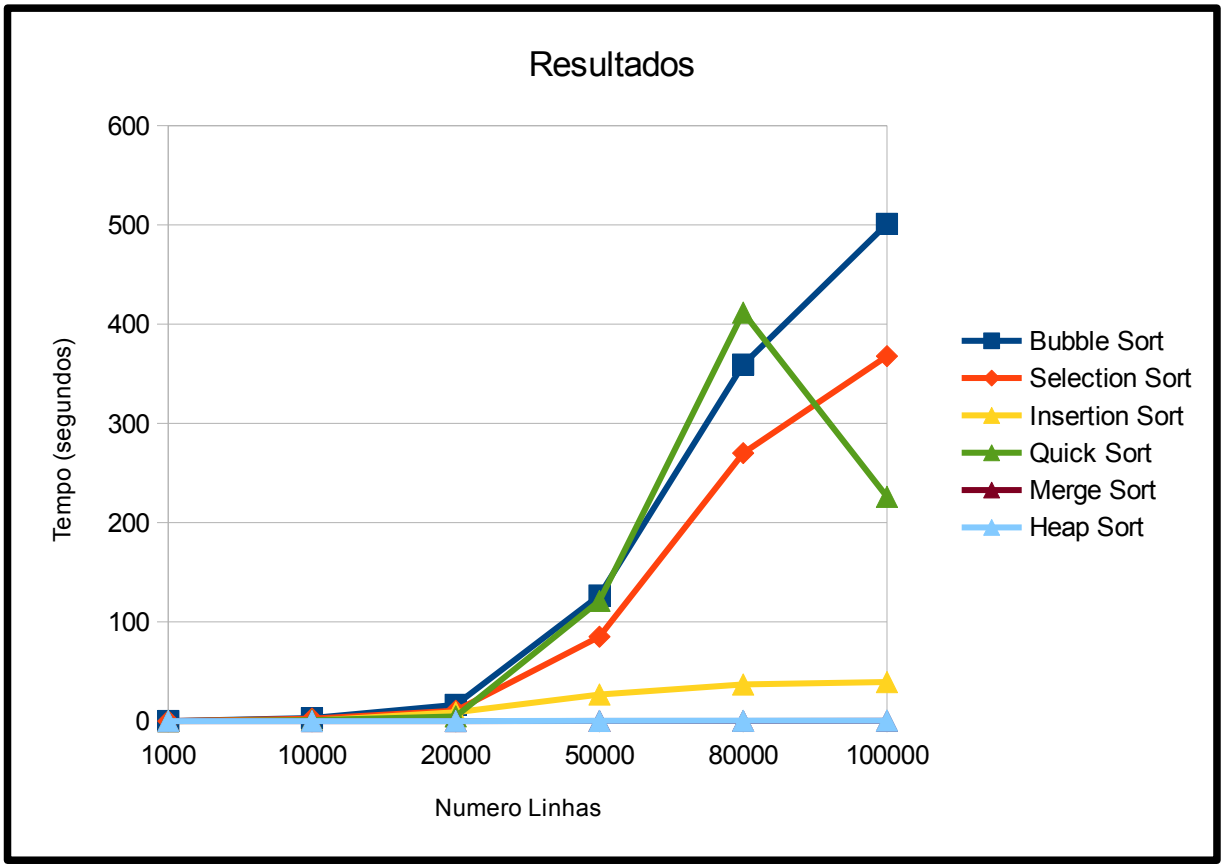
1º Caso – Executando apenas um tipo de ordenação por vez:

Para o caso a seguir foram selecionados de forma aleatória as quantidades de linhas do arquivo de dados em 1.000, 10.000, 20.000, 50.000, 80.000 e 100.000 linhas.

quantidade de linhas/ Tipo ordenação	**1000**	**10000**	**20000**	**50000**	**80000**	**100000**
Bubble Sort	0,0423	3,0826	16,3447	126,5078	359,1927	500,9621
Selection Sort	0,0335	2,4556	10,4417	84,9654	269,9864	367,7091
Insertion Sort	0,0046	0,3256	8,7969	26,7719	36,8551	39,2994
Quick Sort	0,0156	0,9816	4,7243	121,0911	411,5893	225,9566
Merge Sort	0,0017	0,0124	0,027	0,0793	0,1366	0,1791

Heap Sort	0,0024	0,023	0,0629	0,183	0,3331	0,4367

Segue abaixo um gráfico comparativo das ordenações para as quantidades de linhas selecionadas aleatoriamente.



Na condição de dados em ordem aleatória, o Bubble Sort obteve o pior desempenho para ordenar 100 mil linhas, o Quick sort teve o pior desempenho com 80 mil linhas. O Selection Sort teve um desempenho linear a mediada que a quantidade de linhas aumentava e o Insert Sort teve desempenho parecido entre 50 mil e 100 mil linhas de ordenação.

Em relação aos testes:

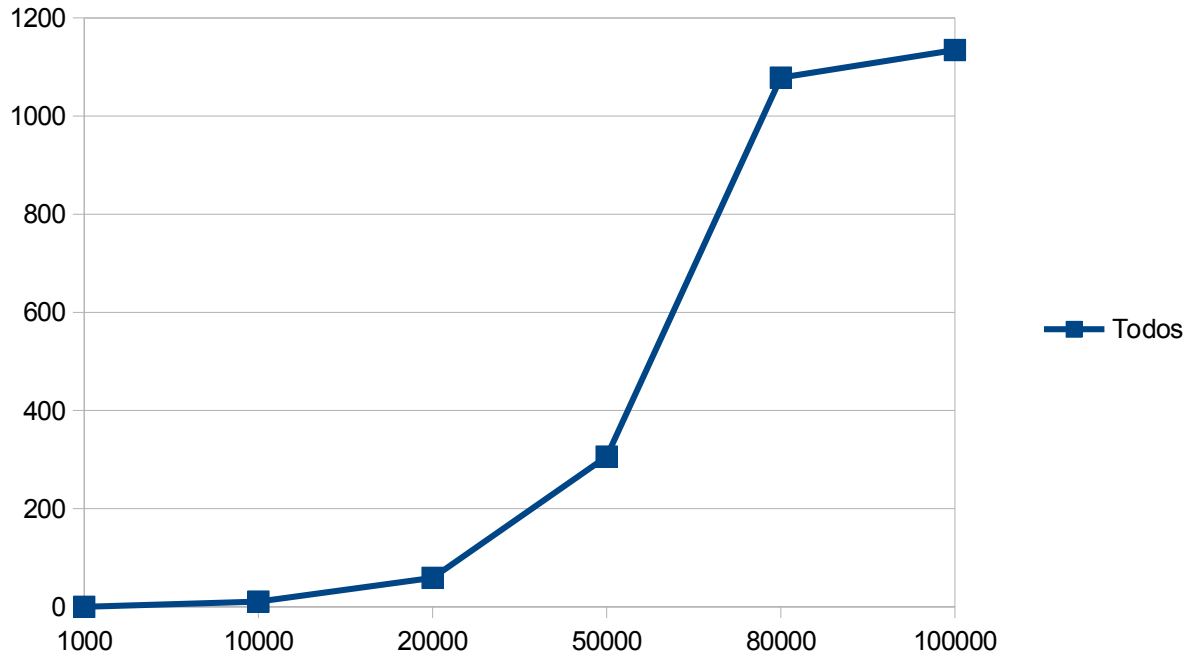
O Heap sort e o Merge sort foram os que apresentaram os melhores desempenhos bem acima dos demais, para todos os tamanhos e tipos de dados experimentados em relação aos outros.

2° Caso – Executando todos os tipos de organizações possíveis

Para o caso a seguir foram selecionados de forma aleatória as quantidades de linhas do arquivo de dados em 1.000, 10.000, 20.000, 50.000, 80.000 e 100.000 linhas.

Tipo ordenação	1000	10000	20000	50000	80000	100000
Todos tipos	0,1507	10,6099	59,0502	306,1041	1078,0932	1134,543

Segue abaixo um gráfico de evolução de todas as ordenações para as quantidades de linhas selecionadas aleatoriamente.



Abaixo segue o Ranking das execuções de todos os tipos de ordenação para 1.000 linhas:

1º 0,0009 segundos Merge Sort

2º	0,0016 segundos	Heap Sort
3º	0,0032 segundos	Insertion Sort
4º	0,0092 segundos	Quick Sort
5º	0,0270 segundos	Selection Sort
6º	0,0343 segundos	Bubble Sort

Abaixo segue o Ranking das execuções de todos os tipos de ordenação para 10.000 linhas:

1º	0,0124 segundos	Merge Sort
2º	0,0230 segundos	Heap Sort
3º	0,3256 segundos	Insertion Sort
4º	0,9816 segundos	Quick Sort
5º	2,4556 segundos	Selection Sort
6º	3,0826 segundos	Bubble Sort

Abaixo segue o Ranking das execuções de todos os tipos de ordenação para 20.000 linhas:

1º	0,0270 segundos	Merge Sort
2º	0,0629 segundos	Heap Sort
3º	4,7243 segundos	Quick Sort
4º	8,7969 segundos	Insertion Sort
5º	10,4417 segundos	Selection Sort
6º	16,3447 segundos	Bubble Sort

Abaixo segue o Ranking das execuções de todos os tipos de ordenação para 50.000 linhas:

1º	0,0793 segundos	Merge Sort
2º	0,1830 segundos	Heap Sort
3º	26,7719 segundos	Insertion Sort
4º	84,9654 segundos	Selection Sort
5º	121,0911 segundos	Quick Sort
6º	126,5078 segundos	Bubble Sort

Abaixo segue o Ranking das execuções de todos os tipos de ordenação para 80.000 linhas:

1º	0,1366 segundos	Merge Sort
2º	0,3331 segundos	Heap Sort
3º	36,8551 segundos	Insertion Sort
4º	269,9864 segundos	Selection Sort
5º	359,1927 segundos	Bubble Sort
6º	411,5893 segundos	Quick Sort

Abaixo segue o Ranking das execuções de todos os tipos de ordenação para 100.000 linhas:

1º	0,1791 segundos	Merge Sort
2º	0,4367 segundos	Heap Sort
3º	39,2994 segundos	Insertion Sort
4º	225,9566 segundos	Quick Sort
5º	367,7091 segundos	Selection Sort

6º	500,9621 segundos	Bubble Sort

5. Considerações Finais

No trabalho foram apresentados o que são métodos de ordenação, alguns dos mais conhecidos, foi explicado como funcionam, seus desempenhos e foram também aplicados na prática em um programa. Em relação aos desempenhos dos métodos, o quick sort é o algoritmo mais eficiente que existe para uma grande variedade de situações, é recursivo, o que demanda uma pequena quantidade de memória adicional.

6. Referências bibliográfico:

https://pt.wikipedia.org/wiki/Algoritmo

https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261

https://dicasdeprogramacao.com.br/o-que-e-algoritmo/

https://brasil.elpais.com/brasil/2018/03/30/tecnologia/1522424604_741609.html

https://www.techtudo.com.br/listas/2020/05/o-que-e-algoritmo-entenda-como-funciona-em-apps-e-sites-da-internet.ghtml

http://www.testonline.com.br/historia-do-algoritmo-os-primeiros-passos-da-computacao/

https://www.alphacodingskills.com/php/pages/php-program-for-bubble-sort.php

https://www.alphacodingskills.com/php/pages/php-program-for-selection-sort.php

https://www.alphacodingskills.com/php/pages/php-program-for-insertion-sort.php

https://www.alphacodingskills.com/php/pages/php-program-for-quick-sort.php

https://www.alphacodingskills.com/php/pages/php-program-for-merge-sort.php

https://www.alphacodingskills.com/php/pages/php-program-for-heap-sort.php

https://pt.wikipedia.org/wiki/Merge_sort

7. Código Fonte

execute_shorts.php

```
<?php
require "public_functions.php";
usleep(mt_rand(100, 10000));
define("PADRAO","\033[0m");
define("VERMELHO","\033[31m");
define("VERDE","\033[32m");
define("AMARELO","\033[33m");
define("AZUL","\033[34m");
define("MAGENTA","\033[35m");
define("CIANO","\033[36m");
define("BRANCO","\033[37m");
$linhas = 0;
$continuar = 's';
$sort = 0;
while (strcasecmp ($continuar,"s") == 0 ) {
    while ($linhas == 0 ) {
        $linhas = readline("Digite o nº de linhas que serão lidas do arquivo : \n");
        if ($linhas <=0){
            echo "você informou um nº de linhas inválido!\n";
            $linhas = 0;
        }
    }
```

```
while ($sort == 0 ) {
 $sort = readline("Digite o tipo de organização que será(ão) realizado(s) :  1 - Buble Sort, 2 - Selection Sort, 3 - Insertion Sort,
4 - Quick Sort, 5 - Merge Sort, 6 - Heap Sort, 7 - Todos: \n");
   if ($sort<=0 || $sort > 7){
      echo "você informou uma opção inválida!\n";
      $sort = 0;
   }
}
$results = array();
switch ($sort) {
   case '1': //Bubble Sort
      echo "iniciando o Bubble Sort...\n";
      $csvData1 = readCSV("metadados_APS.csv",$linhas);
      $time_start1 = microtime(true);
      bubblesort($csvData1,sizeof($csvData1));
      $time_end1 = microtime(true);
       $execution_time1 = number_format((double)($time_end1 - $time_start1), 4, '.', '');
                  array_push($results,array('name' => 'Bubble Sort','value' => $execution_time1,));
      echo "Tempo total de execução de ".$linhas." linhas após o".VERDE." bubble sort".PADRAO." foi de : ".AZUL.$execution_time1.PADRAO." segundos \n";
      break;
   case '2'://Selection Sort
      echo "\niniciando o Selection Sort...\n";
```

```
            $csvData2 = readCSV("metadados_APS.csv",$linhas);
            $time_start2 = microtime(true);
            selectionsort($csvData2,sizeof($csvData2));
            $time_end2 = microtime(true);
              $execution_time2 = number_format((double)($time_end2 - $time_start2), 4,
'.', '');
                        array_push($results,array('name' => 'Selection Sort','value' =>
$execution_time2,));
                echo "Tempo total de execução de ".$linhas." linhas após o".VERDE."
Selection sort".PADRAO." foi de : ".AZUL.$execution_time2.PADRAO." segundos \n";
            break;
        case '3': //Insertion Sort
            echo "\niniciando o Insertion Sort...\n";
            $csvData3 = readCSV("metadados_APS.csv",$linhas);
            $time_start3 = microtime(true);
            insertionsort($csvData3,sizeof($csvData3));
            $time_end3 = microtime(true);
              $execution_time3 = number_format((double)($time_end3 - $time_start3), 4,
'.', '');
                        array_push($results,array('name' => 'Insertion Sort','value' =>
$execution_time3,));
                echo "Tempo total de execução de ".$linhas." linhas após o".VERDE."
Insertion sort".PADRAO." foi de : ".AZUL.$execution_time3.PADRAO." segundos \n";
            break;
        case '4'://Quick Sort
            echo "\niniciando o Quick Sort...\n";
```

```
                $csvData4 = readCSV("metadados_APS.csv",$linhas);
                $time_start4 = microtime(true);
                quicksort($csvData4,0,sizeof($csvData4)-1);
                $time_end4 = microtime(true);
                  $execution_time4 = number_format((double)($time_end4 - $time_start4), 4,
'.', '');
                                array_push($results,array('name' => 'Quick Sort','value' =>
$execution_time4,));
                 echo "Tempo total de execução de ".$linhas." linhas após o".VERDE." Quick
sort".PADRAO." foi de : ".AZUL.$execution_time4.PADRAO." segundos \n";
                break;
            case '5': //Merge Sort
                echo "\niniciando o Merge Sort...\n";
                $csvData5 = readCSV("metadados_APS.csv",$linhas);
                $time_start5 = microtime(true);
                mergesort($csvData5,0,sizeof($csvData5)-1);
                $time_end5 = microtime(true);
                  $execution_time5 = number_format((double)($time_end5 - $time_start5), 4,
'.', '');
                               array_push($results,array('name' => 'Merge Sort','value' =>
$execution_time5,));
                 echo "Tempo total de execução de ".$linhas." linhas após o".VERDE." Merge
sort".PADRAO." foi de : ".AZUL.$execution_time5.PADRAO." segundos \n";
                break;
            case '6':
                //Heap Sort
```

```
                echo "\niniciando o Heap Sort...\n";
                $csvData6 = readCSV("metadados_APS.csv",$linhas);
                $time_start6 = microtime(true);
                heapsort($csvData6,sizeof($csvData6));
                $time_end6 = microtime(true);
                  $execution_time6 = number_format((double)($time_end6 - $time_start6), 4,
'.', '');
                                array_push($results,array('name' => 'Heap Sort','value' =>
$execution_time6,));
                 echo "Tempo total de execução de ".$linhas." linhas após o".VERDE." Heap
sort".PADRAO." foi de : ".AZUL.$execution_time6.PADRAO." segundos \n";
                break;
            case '7':
                echo "iniciando o Bubble Sort...\n";
                $csvData1 = readCSV("metadados_APS.csv",$linhas);
                $time_start1 = microtime(true);
                bubblesort($csvData1,sizeof($csvData1));
                $time_end1 = microtime(true);
                  $execution_time1 = number_format((double)($time_end1 - $time_start1), 4,
'.', '');
                               array_push($results,array('name' => 'Bubble Sort','value' =>
$execution_time1,));
                echo "\niniciando o Selection Sort...\n";
                $csvData2 = readCSV("metadados_APS.csv",$linhas);
                $time_start2 = microtime(true);
                selectionsort($csvData2,sizeof($csvData2));
```

```
$time_end2 = microtime(true);
  $execution_time2 = number_format((double)($time_end2 - $time_start2), 4,
'.', '');
                array_push($results,array('name' => 'Selection Sort','value' =>
$execution_time2,));
$time_start3 = microtime(true);
insertionsort($csvData3,sizeof($csvData3));
$time_end3 = microtime(true);
  $execution_time3 = number_format((double)($time_end3 - $time_start3), 4,
'.', '');
                array_push($results,array('name' => 'Insertion Sort','value' =>
$execution_time3,));
$time_start4 = microtime(true);
quicksort($csvData4,0,sizeof($csvData4)-1);
$time_end4 = microtime(true);
  $execution_time4 = number_format((double)($time_end4 - $time_start4), 4,
'.', '');
                array_push($results,array('name' => 'Quick Sort','value' =>
$execution_time4,));
$time_start5 = microtime(true);
```

```
                mergesort($csvData5,0,sizeof($csvData5)-1);
                $time_end5 = microtime(true);
                  $execution_time5 = number_format((double)($time_end5 - $time_start5), 4,
'.', '');
                                array_push($results,array('name' => 'Merge Sort','value' =>
$execution_time5,));
                echo "\niniciando o Heap Sort...\n";
                $csvData6 = readCSV("metadados_APS.csv",$linhas);
                $time_start6 = microtime(true);
                heapsort($csvData6,sizeof($csvData6));
                $time_end6 = microtime(true);
                  $execution_time6 = number_format((double)($time_end6 - $time_start6), 4,
'.', '');
                                 array_push($results,array('name' => 'Heap Sort','value' =>
$execution_time6,));
                break;
            }

            $total = 0;
          if ($sort==7){
            foreach ($results as $key => $row) {
                $name[$key]  = $row['name'];
                $value[$key] = $row['value'];
            }
            array_multisort($value, SORT_ASC, $results);
            echo "\n\nRanking das execuções: \n";
```

```
    for ($i = 0; $i < sizeof($results); $i++) {
        echo ($i+1)."º : ".VERMELHO.$results[$i]['value']." segundos ".PADRAO."\t";
        echo " -\t".AZUL.$results[$i]['name'].PADRAO."\n";
        $total = $total + $results[$i]['value'];
    }
echo "Tempo total de todos: ".$total;
}
echo "\n\n";
$continuar = (string) readline('Deseja continuar? (S/N): ');
$linhas = 0;
$sort = 0;
}
?>
```

public_functions.php

```
<?php
function readCSV($file,$limit=-1)
{
  $row      = 0;
  $csvArray = array();
  if( ( $handle = fopen($file, "r") ) !== FALSE ) {
    while( ( $data = fgetcsv($handle, 0, ";") ) !== FALSE ) {
      $num = count($data);
      for( $c = 0; $c < $num; $c++ ) {
        $csvArray[$row][] = $data[$c];
      }
```

```
      if ($row==$limit){break;}
      $row++;
    }
  }
  if( !empty( $csvArray ) ) {
    return array_splice($csvArray, 1);
  } else {
    return false;
  }
}
function bubblesort(&$Array, $n) {
    $temp;
    for($i=0; $i<$n; $i++) {
      for($j=0; $j<$n-$i-1; $j++) {
        if($Array[$j]>$Array[$j+1]) {
          $temp = $Array[$j];
          $Array[$j] = $Array[$j+1];
          $Array[$j+1] = $temp;
        }
      }
    }
  }
  function selectionsort(&$Array, $n) {
    for($i=0; $i<$n; $i++) {
      $min_idx = $i;
```

```
    for($j=$i+1; $j<$n; $j++) {
      if($Array[$j] < $Array[$min_idx])
      {$min_idx = $j;}
    }
    $temp = $Array[$min_idx];
    $Array[$min_idx] = $Array[$i];
    $Array[$i] = $temp;
  }
}
function insertionsort(&$Array, $n) {
  for($i=0; $i<$n; $i++) {
    $curr = $Array[$i];
    $j = $i - 1;
    while($j >= 0 && $curr < $Array[$j]) {
      $Array[$j + 1] = $Array[$j];
      $Array[$j] = $curr;
      $j = $j - 1;
    }
  }
}
function quicksort(&$Array, $left, $right) {
  if ($left < $right) {
    $pivot = partition($Array, $left, $right);
    quicksort($Array, $left, $pivot-1);
```

```
    quicksort($Array, $pivot+1, $right);
   }
  }
  function partition(&$Array, $left, $right) {
   $i = $left;
   $pivot = $Array[$right];
   for($j = $left; $j <=$right; $j++) {
    if($Array[$j] < $pivot) {
     $temp = $Array[$i];
     $Array[$i] = $Array[$j];
     $Array[$j] = $temp;
     $i++;
    }
   }
   $temp = $Array[$right];
   $Array[$right] = $Array[$i];
   $Array[$i] = $temp;
   return $i;
  }
function mergesort(&$Array, $left, $right) {
   if ($left < $right) {
    $mid = $left + (int)(($right - $left)/2);
    mergesort($Array, $left, $mid);
    mergesort($Array, $mid+1, $right);
    merge($Array, $left, $mid, $right);
```

```
  }
}
function merge(&$Array, $left, $mid, $right) {
  $n1 = $mid - $left + 1;
  $n2 = $right - $mid;
  $LeftArray = array_fill(0, $n1, 0);
  $RightArray = array_fill(0, $n2, 0);
  for($i=0; $i < $n1; $i++) {
    $LeftArray[$i] = $Array[$left + $i];
  }
  for($i=0; $i < $n2; $i++) {
    $RightArray[$i] = $Array[$mid + $i + 1];
  }
  $x=0; $y=0; $z=$left;
  while($x < $n1 && $y < $n2) {
    if($LeftArray[$x] < $RightArray[$y]) {
      $Array[$z] = $LeftArray[$x];
      $x++;
    }
    else {
      $Array[$z] = $RightArray[$y];
      $y++;
    }
    $z++;
  }
```

```
    while($x < $n1) {
      $Array[$z] = $LeftArray[$x];
      $x++;
      $z++;
    }
    while($y < $n2) {
      $Array[$z] = $RightArray[$y];
      $y++;
      $z++;
    }
  }
function heapsort(&$Array, $n) {
    for($i = (int)($n/2); $i >= 0; $i--) {
      heapify($Array, $n-1, $i);
    }
    for($i = $n - 1; $i >= 0; $i--) {
      $temp = $Array[$i];
      $Array[$i] = $Array[0];
      $Array[0] = $temp;
      heapify($Array, $i-1, 0);
    }
  }
  function heapify(&$Array, $n, $i) {
    $max = $i;
    $left = 2*$i + 1;
```

```
    $right = 2*$i + 2;
    if($left <= $n && $Array[$left] > $Array[$max]) {
      $max = $left;
    }
    if($right <= $n && $Array[$right] > $Array[$max]) {
      $max = $right;
    }
    if($max != $i) {
      $temp = $Array[$i];
      $Array[$i] = $Array[$max];
      $Array[$max] = $temp;
      heapify($Array, $n, $max);
    }
  }
?>
```

Ficha de atividades práticas supervisionadas

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Rodrigo Joaquim Sousa de Lima

RA: F321EA0 CURSO: Ciência da Computação

CAMPUS: Brasília SEMESTRE: 9º semestre TURNO: Matutino

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ALUNO	PROFESSOR
2/11	Planejamento	2h	Rodrigo	
3/11	Pesquisa para documentação	4h	Rodrigo	
4/11	Desenvolvimento do referencial teórico	8h	Rodrigo	
5/11	Pesquisa inicial para o código	5h	Rodrigo	
6/11	Estruturação do código	5h	Rodrigo	
7/11	Pesquisa para melhoria do código	5h	Rodrigo	
8/11	Desenvolvimento do código	8h	Rodrigo	
9/11	Pesquisa para melhoria do código	4h	Rodrigo	
10/11	Separação do código em funções	8h	Rodrigo	
11/11	Testes e avaliações do código	8h	Rodrigo	
12/11	Finalização do referencial teórico	8h	Rodrigo	
16/11	Verificação do trabalho	8h	Rodrigo	
18/11	Execução dos testes finais e apresentação dos resultados	10h	Rodrigo	

TOTAL DE HORAS: 83h